Instituto Politécnico Nacional

Escuela Superior De Computo

Sistemas en Chip

Práctica 2

Nombre de los integrantes:

García Quiroz Gustavo Iván

Bejarano García Owen Uriel

Grupo: 7CV3

Nombre Del Profesor: Miguel Ángel Castillo Martínez

Fecha De Entrega: 03/06/2025

1 Índice

2 Introducción

En microcontroladores, la medición del tiempo es fundamental para aplicaciones como generación de señales PWM, control de motores, comunicación serial y sistemas de temporización. Los timers (temporizadores) son periféricos clave que permiten ejecutar tareas periódicas sin depender de retardos por software, lo que optimiza el rendimiento del sistema.

En esta práctica, se exploró el Timer0 del ATmega328P (integrado en Arduino Uno), un temporizador de 8 bits configurable en distintos modos de operación. El objetivo principal fue:

1. Verificar el comportamiento del Timer0 en modo CTC (Clear Timer on Compare Match), utilizando el registro OCR0A y un prescaler para generar una frecuencia específica.
2. Sintetizar una señal de 1 kHz con error mínimo, aplicando criterios de selección de prescaler y valor de comparación (OCR0A) para minimizar discrepancias entre el cálculo teórico y la implementación real.

Esta práctica no solo refuerza el entendimiento de los registros de control del timer (TCCR0A, TCCR0B, OCR0A), sino que también introduce el uso de herramientas de diagnóstico como el osciloscopio, esencial para validar señales generadas por hardware en sistemas.

3 Marco teórico

3.1 Temporizadores en Microcontroladores

Los temporizadores (timers) son módulos esenciales en los microcontroladores, diseñados para medir intervalos de tiempo, generar retardos o producir señales periódicas sin necesidad de bloquear la ejecución del programa principal. Funcionan mediante un contador que incrementa su valor a una frecuencia determinada por el reloj del sistema y un prescaler, que divide esta frecuencia para ajustar la velocidad de conteo.

En los sistemas embebidos como Arduino, los timers permiten implementar funciones críticas como:

- Generación de señales PWM (Pulse Width Modulation).

- Control de tiempos en comunicaciones seriales.

- Temporización de eventos en sistemas en tiempo real.

3.2 Timer0 en Arduino UNO

El Arduino UNO utiliza el microcontrolador ATmega328P, que incluye tres temporizadores: Timer0, Timer1 y Timer2. El Timer0 es un temporizador de 8 bits compartido con funciones clave como delay() y millis(). Sus características principales incluyen:

- Modos de operación:
    - Normal: El contador incrementa hasta 255 y vuelve a 0.
    - CTC (Clear Timer on Compare Match): El contador se reinicia al alcanzar el valor almacenado en el registro OCR0A, permitiendo generar frecuencias precisas.
    - Fuente de reloj: Puede usar el reloj interno (16 MHz en Arduino) con distintos prescalers (1, 8, 64, 256, 1024).

3.3 Modo CTC y Frecuencia de Salida

En el modo CTC, la frecuencia de la señal generada en el pin OC0A (pin 6 en Arduino UNO) se calcula con la fórmula:

$$f_{generada} = \frac{f_{osc}}{2 \times prescaler \times (OCR0A + 1)}$$

Donde:

f_{osc}: Frecuencia del reloj (16 MHz para Arduino).

OCR0A: Valor del registro de comparación (0 a 255).

prescaler: Divisor de frecuencia (ej: 64, 1024).

Este modo es ideal para generar señales periódicas con bajo error, ya que el contador se reinicia automáticamente al coincidir con OCR0A.

3.4 Registros del Timer0

La configuración del Timer0 se realiza mediante tres registros principales:

TCCR0A (Timer/Counter Control Register A):

- Bits WGM01:WGM00: Seleccionan el modo (CTC = 01).
- Bits COM0A1:COM0A0: Configuran la salida (toggle = 01).

TCCR0B (Timer/Counter Control Register B):

- Bits CS02:CS00: Seleccionan el prescaler (ej: 1024 = 101).

OCR0A (Output Compare Register A):

- Define el valor máximo del contador en modo CTC.

4 Materiales y Equipo Utilizado

Material/Equipo	Especificaciones
Arduino Uno	Microcontrolador ATmega328P, 16 MHz
Osciloscopio Rigol	Modelo DS1054Z (4 canales)
Cables de conexión	Jumpers MM y BNC caimán
Computadora	Con IDE Arduino instalado

5 Desarrollo de la Práctica

En esta sección se detallan las dos pruebas realizadas con el Timer0 del Arduino Uno, explicando los cálculos teóricos, la configuración de registros y los resultados obtenidos mediante el osciloscopio.

5.1 Prueba 1: Verificación de frecuencia con Timer0 (8 bits)

El objetivo fue comprobar que la frecuencia generada en el pin OC0A (pin 6) coincide con el cálculo teórico al configurar el Timer0 en modo CTC (*Clear Timer on Compare Match*), usando un valor específico de OCR0A y un prescaler de 1024.

Cálculos:

- **Frecuencia del reloj (CPU):** 16 MHz.
- **Modo del Timer0:** CTC (configuración de registros TCCR0A y TCCR0B).
- **Parámetros clave:**
  - OCR0A = 147.
  - Prescaler = 1024.
- **Fórmula aplicada:**

$$fgenerada = fosc2 \times prescaler \times (OCR0A + 1) fgenerada$$
$$= 2 \times prescaler \times (OCR0A + 1) fosc$$

Sustituyendo valores:

$$f = 16\,000\,0002 \times 1024 \times (147 + 1) \approx 52.78\,Hz. f$$
$$= 2 \times 1024 \times (147 + 1) 16000000 \approx 52.78 Hz.$$

Implementación en código:

```c
#include <stdint.h>

typedef struct {
    uint8_t b0 : 1;
    uint8_t b1 : 1;
    uint8_t b2 : 1;
    uint8_t b3 : 1;
    uint8_t b4 : 1;
    uint8_t b5 : 1;
    uint8_t b6 : 1;
    uint8_t b7 : 1;
} bitfield_t;

typedef union {
    bitfield_t bits;
    uint8_t value;
} reg8_t;

// Definición de punteros a registros del Timer0 (en ATmega328P)
volatile reg8_t *my_TCCR0A = (reg8_t *)0x44;  // Registro de control A del Timer0
volatile reg8_t *my_TCCR0B = (reg8_t *)0x45;  // Registro de control B del Timer0
volatile reg8_t *my_OCR0A  = (reg8_t *)0x47;  // Registro de comparación A
volatile reg8_t *my_DDRD   = (reg8_t *)0x2A;  // Registro de dirección de datos D (OC0A = PD6)
```

```c
int main() {
    // Configurar PD6 (OC0A) como salida (pin 6 en Arduino Uno)
    my_DDRD->bits.b6 = 1;

    // Configurar Timer0 en modo CTC (WGM02=0, WGM01=1, WGM00=0)
    my_TCCR0A->value = (1 << 1);  // Modo CTC (WGM01=1)
    my_TCCR0A->bits.b6 = 1;        // Toggle OC0A en comparación (COM0A0=1)

    // Asignar valor de OCR0A = 147
    my_OCR0A->value = 147;

    // Configurar prescaler = 1024 (CS02=1, CS01=0, CS00=1)
    my_TCCR0B->value = (1 << 2) | (1 << 0);

    while (1) {
        // Nada que hacer aquí, el Timer0 genera la señal automáticamente
    }

    return 0;
}
```

Figura 1 Verificación de frecuencia con Timer0 (8 bits).

Resultados y verificación:

- **Señal observada en el osciloscopio:** Una onda cuadrada con frecuencia cercana a 52.78 Hz.

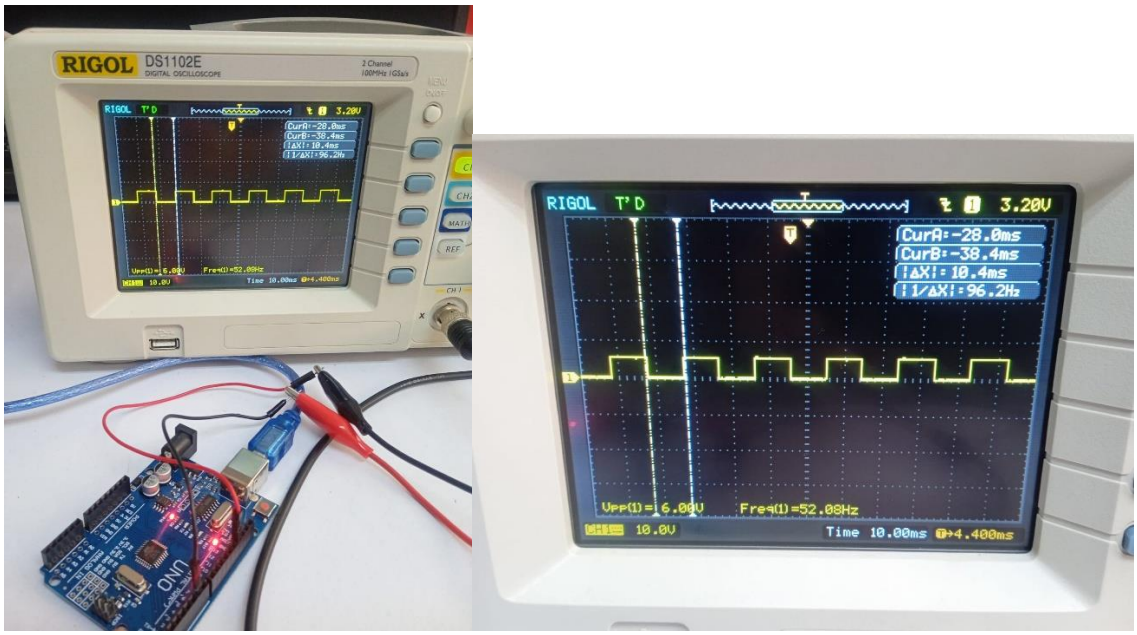

Figura 2. Onda cuadrada con frecuencia de 52.08

5.2 Prueba 2: Generación de 1 kHz con error mínimo

El objetivo fue configurar el Timer0 para generar una señal de 1 kHz exactos en el pin OC0A, minimizando el error mediante la selección adecuada del prescaler y el valor de OCR0A.

Parámetros:

1. **Prueba con distintos prescalers:**
    - Se evaluaron los prescalers disponibles (1, 8, 64, 256, 1024) para encontrar una combinación que permita un OCR0A entero.
    - **Prescaler = 64** fue el único que permitió un valor exacto:

$$OCR0A = \frac{16\,000\,000}{2 \times 64 \times 1000} - 1 = 124$$

    - **Frecuencia resultante:**

$$f = \frac{16\,000\,000}{2 \times 64 \times (124 + 1)} = 1000\,Hz$$

Implementación en código:

```c
#include <stdint.h>

typedef struct {
    uint8_t b0 : 1;
    uint8_t b1 : 1;
    uint8_t b2 : 1;
    uint8_t b3 : 1;
    uint8_t b4 : 1;
    uint8_t b5 : 1;
    uint8_t b6 : 1;
    uint8_t b7 : 1;
} bitfield_t;

typedef union {
    bitfield_t bits;
    uint8_t value;
} reg8_t;

// Definición de punteros a registros del Timer0
volatile reg8_t *my_TCCR0A = (reg8_t *)0x44;  // Registro de control A del Timer0
volatile reg8_t *my_TCCR0B = (reg8_t *)0x45;  // Registro de control B del Timer0
volatile reg8_t *my_OCR0A  = (reg8_t *)0x47;  // Registro de comparación A
volatile reg8_t *my_DDRD   = (reg8_t *)0x2A;  // Registro de dirección de datos D (OC0A = PD6)
```

```c
int main() {
    // Configurar PD6 (OC0A) como salida (pin 6 en Arduino Uno)
    my_DDRD->bits.b6 = 1;

    // Configurar Timer0 en modo CTC (WGM02=0, WGM01=1, WGM00=0)
    my_TCCR0A->value = (1 << 1);  // Modo CTC (WGM01=1)
    my_TCCR0A->bits.b6 = 1;       // Toggle OC0A en comparación (COM0A0=1)

    // Asignar valor de OCR0A = 124 para 1 kHz (con prescaler=64)
    my_OCR0A->value = 124;

    // Configurar prescaler = 64 (CS02=0, CS01=1, CS00=1)
    my_TCCR0B->value = (1 << 1) | (1 << 0);

    while (1) {
        // Nada que hacer aquí, el Timer0 genera la señal automáticamente
    }

    return 0;
}
```

Figura 3 Generación de 1 kHz con error mínimo.

Resultados y verificación:

- **Señal en el osciloscopio:** Onda cuadrada de 1.00 kHz (ver captura anexa).

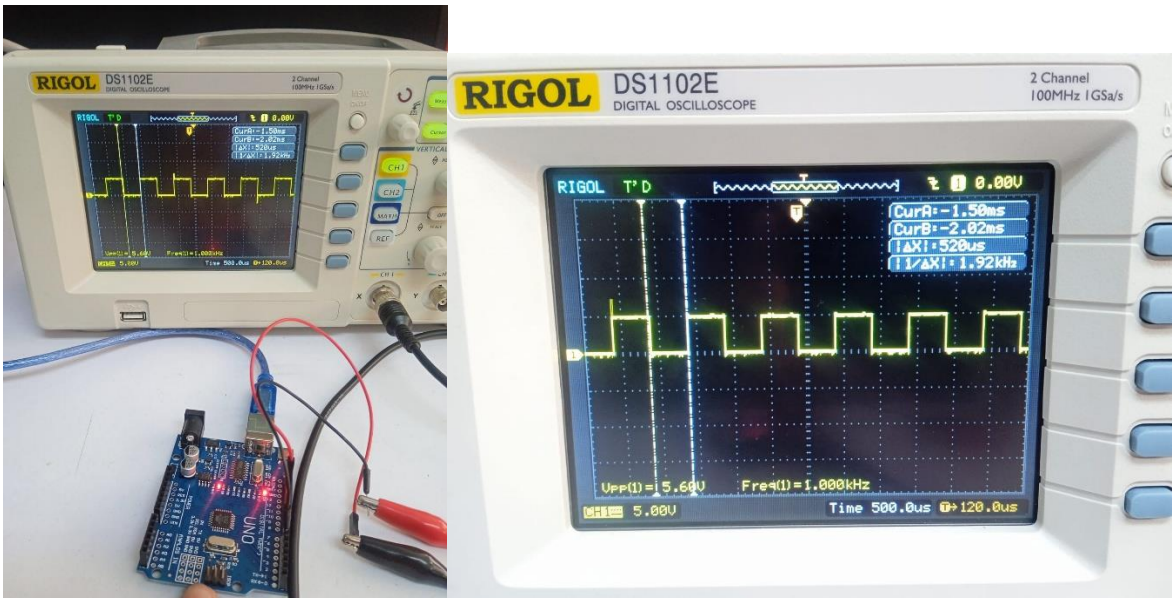

Figura 4. Onda cuadrada con frecuencia de 1 KHz

6 Conclusiones

- Bejarano García Owen Uriel

En esta práctica se logró comprender la importancia de los temporizadores en microcontroladores, específicamente el Timer0 de 8 bits en Arduino, para generar señales de frecuencia precisa. Mediante el modo CTC (Clear Timer on Compare Match), se pudo controlar la frecuencia de salida ajustando el registro OCR0A y el prescaler, demostrando que la selección adecuada de estos valores es clave para minimizar errores.

- García Quiroz Gustavo Iván

En la Prueba 1, al configurar OCR0A = 147 y un prescaler de 1024, se obtuvo una señal de 52.78 Hz, validando que el cálculo teórico coincide con la medición práctica en el osciloscopio. Esto confirmó que, a mayor prescaler, la frecuencia generada es menor, pero con mayor resolución. Por otro lado, en la Prueba 2, se optimizó la configuración para lograr una señal exacta de 1 kHz (OCR0A = 124, prescaler = 64), destacando que no todos los valores de prescaler permiten frecuencias exactas, por lo que es necesario analizar las combinaciones posibles.

7 Referencias

[1] D. Features, “8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash,” Microchip.com. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. [Accessed: 25-Mar-2025].

[2] Rohde and Schwarz International, “Qué es UART,” Rohde-schwarz.com. [Online]. Available: https://www.rohde-schwarz.com/es/productos/test-y-medida/essentials-test-equipment/digital-oscilloscopes/que-es-uart_254524.html. [Accessed: 25-Mar-2025].